Federated or Split? A Performance and Privacy Analysis of Hybrid Split and Federated Learning Architectures

Valeria Turina
Computer Science
Saint Louis University
valeria.turina@slu.edu

Zongshun Zhang
Computer Science
Boston University
zhangzs@bu.edu

Flavio Esposito
Computer Science
Saint Louis University
flavio.esposito@slu.edu

Ibrahim Matta
Computer Science
Boston University
matta@bu.edu

2021 IEEE 14th International Conference on Cloud Computing (CLOUD) | 978-1-6654-0060-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/CLOUD53861.2021.00038

***Abstract*—Mobile phones, wearable devices, and other sensors produce every day a large amount of distributed and sensitive data. Classical machine learning approaches process these large datasets usually on a single machine, training complex models to obtain useful predictions. To better preserve user and data privacy and at the same time guarantee high performance, distributed machine learning techniques such as Federated and Split Learning have been recently proposed. Both of these distributed learning architectures have merits but also drawbacks. In this work, we analyze such tradeoffs and propose a new hybrid Federated Split Learning architecture, to combine the benefits of both in terms of efficiency and privacy. Our evaluation shows how Federated Split Learning may reduce the computational power required for each client running a Federated Learning and enable Split Learning parallelization while maintaining a high prediction accuracy with unbalanced datasets during training. Furthermore, FSL provides a better accuracy-privacy tradeoff in specific privacy approaches compared to Parallel Split Learning.**

***Index Terms*—Split Learning, Federated Learning, Privacy.**

I. Introduction

Several devices today, e.g., mobile phones and wearable technologies, produce a vast amount of data. If processed correctly with machine learning (ML) and deep neural network algorithms, such data can be used to improve revenue, user experience, and even our health.

Historically, it has been sufficient to train deep neural networks over centralized architectures. Recently, however, such a centralized training approach is becoming unsustainable. Applications have been demanding more frequent retraining [1] and the scale of the data to be processed has grown. Moreover, a critical issue of many data-driven applications is privacy. It is not always possible to share datasets with a third party, e.g., a cloud provider.

The limitation of centralized learning techniques has led to the design of several distributed machine learning architectures [2]–[4], the first one being Federated Learning [4]. The design principle of such distributed learning architectures assumes that the same (deep) learning model is sent to the clients that have the data instead of sending the data to the model as in centralized learning architectures. Each client possesses or receives only a partition of the data though. Subsequently, the different neural network weights, i.e., parameters, of each client are exchanged during the training phase iteratively until the model converges. All clients send their weights to a Parameter Server at each iteration to then receive back an average of such weights.

Distributed learning in general and Federated Learning, in particular, became a necessity for many applications, such as Natural Language Processing, e.g., on Google's Gboard mobile keyboard [5], [6], voice recognition for Apple's Siri, or keyword spotting [7]. In other applications, however, applying federated learning is challenging: the clients required to run the sub-training process may not have enough computational capacity to train the entire neural network model, and the storage may be insufficient for this goal. Finally, the privacy of the data may be violated because the neural network's weights are being sent over the network during the training, exposing the model to machine learning inference attacks [8]. To partially overcome these limitations, recently, a new distributed learning paradigm — Split Learning [2] — arose. In Split Learning, multiple agents systematically split the deep neural network (composed of many computational layers) and run a subset of these layers, sharing over the network only the intermediate data, i.e., the activation results of the last layer in clients. *SplitNN* lowered the computing power required at each device to train the neural network with respect to a classical federated learning model. By offloading the first subset of layers to each client, this method also prevents source data from being sent over a network, hence (at least partially) limiting data-sharing concerns. However, the challenge of this type of distributed learning is that processes need to run sequentially by design and hence are hard to parallelize. Furthermore, the training process may not reach convergence sharply or may not reach convergence at all if the dataset is unbalanced [9] [10].

In this work, we study how to improve the efficiency and privacy of Split and Federated learning by proposing a new learning architecture that combines both approaches. We call this architecture *Federated Split Learning*. By merging the two architectures, we can tune the neural network's weights or data that needs to be kept at every single data processing server or device, due to computation capacity or privacy requirements, as in Split Learning, but we also exploit the Federated Learning principle of parallelizing the training process.

We compare our learning architecture with the state of art hybrid split-federated architecture — Parallel Split Learning [3] — assessing both performance and (lack of) privacy. We found interesting tradeoffs in tuning the distribution of data available at each client, monitoring the computational time,

2159-6190/21/$31.00 ©2021 IEEE
DOI 10.1109/CLOUD53861.2021.00038

and estimating the level of privacy preserved. In particular, we show that our Federated Split Learning architecture can reach better performance in terms of computational time as it can parallelize the training processes. Moreover, we analyze different approaches to preserve the privacy of this architecture under an inverse engineering *attacker* model. We also propose a Client-Based Privacy Approach to preserve the privacy of the source data.

Our contributions. In this paper, we present three main contributions: (1) We propose a new distributed machine learning architecture called *Federated Split Learning* that is advantageous in terms of efficiency and privacy with respect to the state of art solutions. (2) We propose a general Client-Based Privacy Approach as a countermeasure against machine learning reverse engineering attacks. (3) We evaluate our hybrid learning architecture using both simulations and a prototype. Our simulations run on a single machine using both the Pytorch [11] and the PySyft [12] libraries, while our decentralized prototype implementation uses the PyGrid library [13].

The rest of this paper is organized as follows. In Section II we present some background on split and federated learning and analyze related work dissecting solutions that combine both learning architectures. In Section III, we discuss the details of our *Federated Split Learning* (FSL) architecture. In Section IV, we introduce our generic Client-Based Privacy Approach, and we analyze the privacy guaranteed by our FSL architecture with this enhancement. In Section V, we discuss our evaluation in terms of computation time, memory consumption, and accuracy for varying number of clients. In Section VI we present our privacy evaluation experimental results, comparing our Client-Based Privacy Approach. Finally, in Section VII we conclude the paper.

II. Federated Learning and Split Learning: Background and Related Work

In this section, we give some background on *Split learning (SL)* and *Federated learning (FL)*, highlighting their architecture's advantages and limitations when used in isolation.

In a *Split learning* architecture [2] (Figure 1 left) a Deep Neural Network is partitioned into two (or multiple) parts [14]. Each client runs the first partition of the Deep Neural Network model (a few layers) while the second partition runs on a single server, for example, at the edge of the network. During the training phase, the first client sends the intermediate data to a server in a so-called *forward propagation*; such forward propagation is represented with a solid arrow in Figure 1 (left). During the gradient descent algorithm, the server sends back to the client the value of the gradients (dashed arrow in Figure 1 left) in the corresponding *backward propagation*. [1] Then, the next client starts the training on its data partition as described before, so the architecture runs a sequential algorithm. While this learning architecture has merits [2], it

[1]Gradient Descent is an optimization technique that is used to improve deep learning and neural network-based models by progressively minimizing a cost function.

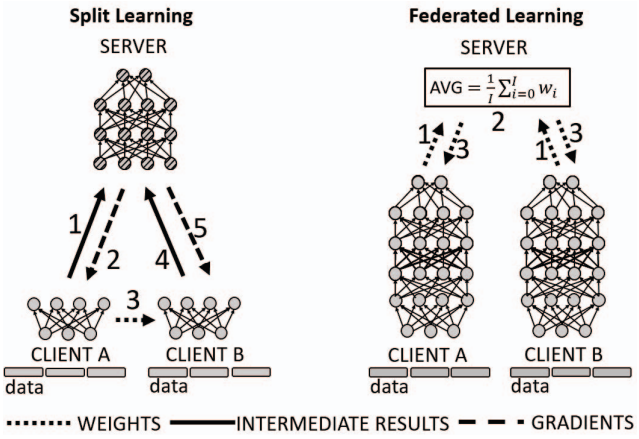

Fig. 1: Layers partitioned in Neural Networks with Split (left) and Federated (right) Learning. The numbers represent the order of processing. The different types of lines show the dataflow between computations.

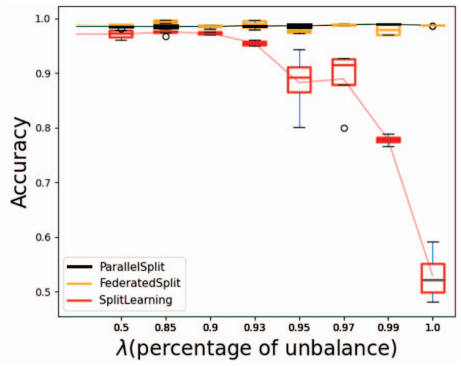

Fig. 2: Drawback of Split Learning: Unbalance of data in each Client and level of accuracy.

suffers from two main drawbacks. First, the training process is sequential, leading to inefficiencies and under-utilization of the computational resources. Second, the model may not converge to an accurate model if the dataset in each Client has unbalanced features (i.e., non-IID).

We illustrate this second drawback with an experiment (Figure 2). In particular, we used the well-known MNIST dataset [15], in which a neural network is trained to recognize hand-written digits. We split such neural network among two clients, with an unbalanced distribution of training set images between the two clients: Client 1 runs the training process with the majority of even digits while Client 2 has more odd digits. We then defined a parameter λ to measure the percentage of data unbalance on each client running the training process. When the parameter λ approaches one, i.e., Client 1 has a larger number of even digits, the accuracy of the Split learning model decreases dramatically (Figure 2), as we expected. This is because the machine learning model attempts to fit on two datasets with very different features (captured by λ), and it fails to converge to a correct single model.

In Federated learning architectures [4] (Figure 1 right), the entire deep learning model is sent to each client, that has an amount of data. After each round, defined in FL, the clients exchange the neural network weights with a server. This so-called "parameter" server then computes and sends back to the clients the average of such weights, and the process is periodically repeated until the decentralized convergence process terminates. An advantage of this architecture is scalability in that each client can run in parallel. However, to train the full model, clients may require computational power that is hardly available at edge sites. Moreover, while the data is local at every edge, the weights shared during the averaging step can be used to reconstruct the source dataset. Prior work has reported on this kind of attack [16] and [17]. We will also use this averaging idea in our approach, but since there are methods to defend against such attack [18], and our approach does not share the weights over the public network, this vulnerability is not present in our system.

Combining Split and Federated Learning. After introducing both architectures separately, we now discuss some existing distributed techniques that merge Split Learning and Federated Learning. We begin with SplitFed [19], a distributed algorithm that combines the ideas of computing the weight average, a characteristic of the Federated Learning architecture, and the neural network split between client and server of the Split Learning architecture. Each client starts the forward propagation phase; the computation results are sent to the server that completes the phase. SplitFed computes the forward propagation of the server's neural network sequentially, choosing randomly the client's output to use. The backward propagation is computed within the server, and then the result is backward-propagated to each client. Before updating the clients' weights, a (secure) server, called FedServer, computes the average of the clients' weights and then sends it back to each client. One of the most notable drawbacks of the SplitFed architecture is that the forward and backward propagation process in the server is not scalable when the number of clients grows. SplitFed assumes a single server or cluster with sufficient capacity, which can become a bottleneck when the number of clients grows. Furthermore, SplitFed may fail to support latency-sensitive applications in WAN environments.

Another attempt to combine SL and FL is called Parallel Split Learning (PSL) [3], whose architecture is shown in Figure 3. Note the similar behavior in both SL and PSL. PSL also splits the deep neural network into two parts. The first part is sent to I different clients and the second to a single server. After an initialization phase, all clients and the single server exchange intermediate data and the gradients during the training process. [2] Unlike a Split Learning architecture, the training phase of PSL is parallelized (see Algorithm 1). Despite such parallelism, PSL relies on only one server, which may become the training process's performance bottleneck. In the worst case, the training may become sequential, assuming significant speed gaps between clients. The server cannot start the backward propagation phase until all clients have completed their forward propagation phase. The problem may be exacerbated as the number of clients scales up. Aside from time bottlenecks, such a "semi-sequential" process at the server may require an unsustainable amount of memory to temporarily store the intermediate data from all clients, especially when the clients are producing training data faster than the server can consume them.

Another interesting architecture that has been proposed is FedSL [20]. FedSL was designed to work with ordered data sequences, such as text or time series. FedSL can train a Machine Learning model such as a Recurrent Neural Network (RNN) in a distributed fashion. Our proposed *Federated Split Learning* architecture, which we describe in the next section, was designed to generalize the FedSL approach by extending its advantages to other ML tasks in which the training data does not need to be an ordered data stream.

Privacy-Aware Centralized and Distributed Learning. Prior work analyzed how to prevent inference on the neural network, considering a model to be private if it does not allow reconstruction or inference of either the dataset used for training or the neural network itself. Despite based on a centralized learning architecture, the closest solution to ours is NoPeek, a recent algorithm introduced in [21]. NoPeek aims at preserving the privacy of the centralized learning architectures. NoPeek adds a privacy increment, called *Distance Correlation (DC)* [21], to the Cross-Entropy. Cross-Entropy is a widely popular loss function in machine learning. Applying NoPeek to distributed settings may result in a loss of privacy gain. This is because we would be forced to expose too much information to the attacker. By design of the NoPeek approach, the Distance Correlation (DC) increment is added to the edge server's loss function. This could be a potential leak of information. For example, it would be readily possible to reconstruct the original data having both the Distance Correlation value and the intermediate data sent over the network to the server during the training process. It would be desirable to force clients and servers to use different loss functions to avoid such privacy leaks.

Another line of work [22] uses the notion of Differential Privacy in learning. For example, the Pytorch Opacus library [23], which we used in our evaluation, adds a Gaussian noise, controlled by parameters like noise multiplier (ϵ), to the neural network gradients to preserve the privacy of the training data. Our implementation enables centralized approaches like NoPeek or Differential Privacy to be deployed in distributed learning architectures.

III. Privacy-Oblivious Federated Split Learning

Motivated by the previously discussed suboptimalities of current distributed learning architectures, in this Section we discuss our proposed *Federated Split Learning (FSL)*, shown in Figure 3 (*Right*). We distinguish three types of nodes in our architecture: clients, edge servers, and parameter servers. We use edge servers to differentiate from the PSL architecture in

[2]The notion of *intermediate data* was introduced in split learning [2] and represents the activation result of the forward propagation inside the client's neural network.

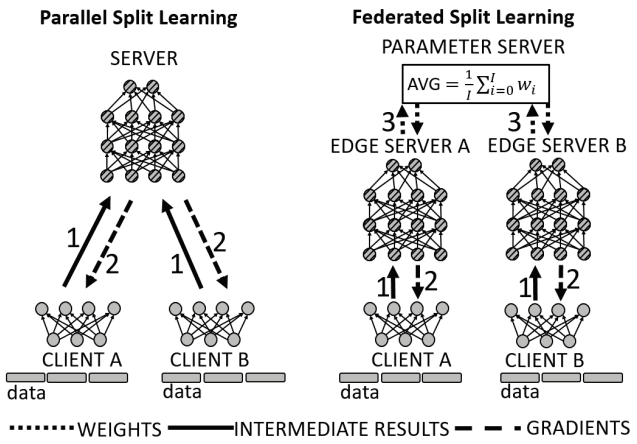

Fig. 3: *Parallel Split Learning* [3] [left]: clients send the intermediate data to a single server (1). It computes the loss function and sends back to clients weights' gradient of the Neural network (2).
Federated Split Learning (our proposal) [right]: each pair of client and edge server exchanges the intermediate data of the neural network and the gradients (1,2); after the last batch of data (end of an epoch) owned by each client is used for training, a parameter server averages the edge servers' weights (3).

which a single logically centralized server is used. In FSL, the deep neural network is logically split into two partitions. The first partition containing the input layer of the learning model is sent to a client, while the second partition runs within an (edge) server.

We start by describing a Privacy-Oblivious Federated Split Learning architecture (Algorithm 2), and we then compare such version with Parallel Split Learning (PSL) [3], a related hybrid architecture detailed in Algorithm 1. In our *Federated Split Learning*, for each pair of client and edge server processes, we start a forward propagation inside the client (line 7 of Algorithm 2). Then the client sends the intermediate data, i.e., the activation results to its corresponding edge server (line 8), and the edge server continues the forward propagation phase (line 9). After the edge server obtains its output, it computes the value of the loss function (line 10) and it starts the backward propagation (lines 11 and 12). When the calculation of the gradients reaches the intermediate data, the gradients are sent from the edge server to the client (line 13). Finally, the privacy-oblivious FSL algorithm terminates the backward propagation in the client with those gradients (line 14). When all pairs finish processing their batches of data, the weights in the edge servers are averaged in a parameter server[3] and then sent back to each edge server (line 17).

We define all those steps starting at all clients as they begin forward propagation until the averaging step finishes as one *epoch*.

Remark. *Most of the steps of Algorithms 1 and 2 are similar; however, a key difference between the two is the interaction between each client and server. In Algorithm 1, each client runs the forward propagation and sends the intermediate data in parallel (lines 8-9). When the server completes processing the outputs of all clients, the gradients are sent to each corresponding client to terminate the backward step (lines 17 and 18).*

[3]In some machine learning library, what we call *parameter server* is also called "worker".

Variable Name	Meaning in Algorithm
L_i	Neural network layer i
B_i	Batch of data in client i
$labels_i$	Labels in client i
$Interm\_r_i$	intermediate data in client i
$Loss_i$	Loss function result in server i(FSL)
$Loss$	Loss function result in server(PSL)
$gradient_i$	loss function gradient in pair i(FSL)
$gradient$	loss function gradient(PSL)
$gradient'_i$	gradient during backward propagation in pair i (FSL)
$gradient'$	gradient during backward propagation (PSL)
DC	$Distance\_Correlation$ function [21]
$frequency$	Frequency to call $Distance\_Correlation$
ϵ	Random Noise (e.g. Laplace)
Function Name	**Meaning in Algorithm**
$F_{c,i}$	Forward propagation in client i
$F_{s,i}$	Forward propagation in edge server i (FSL)
F_s	Forward propagation in edge server (PSL)
$F^T_{c,i}$	Backward propagation in client i
$F^T_{s,i}$	Backward propagation in edge server i (FSL)
F^T_s	Backward propagation in edge server (PSL)
G	Calculate Gradient of Loss function
$Loss\_func_s$	Loss function to be called in each server
$Loss\_func_c$	Loss function to be called in each client
Constant Name	**Meaning in Algorithm**
I	Number of clients
N	Number of Neural Network layers

TABLE I: Notation used in the FSL (Federated Split Learning) and PSL (Parallel Split Learning) algorithms.

Unlike PSL, with our proposed FSL architecture, multiple pairs of clients and edge servers could simultaneously run training or inference processes. Each edge server only needs to work with its client, decoupling the performance dependencies on other clients. Moreover, the network communication overhead caused by the transmissions between a client and its edge server is small since we do not need to transfer the training datasets. Furthermore, since FSL does not require sharing clients' neural network weights, it is more resilient against some inversion attacks. Aside from these benefits, one potential limitation of the Federated Split Learning architecture is the higher overhead with respect to PSL due to the additional weight averaging step (Algorithm 2, line 17).

We have discussed the privacy-oblivious FSL (Algorithm 2). It may be desirable to preserve privacy with respect to the data exchanged between clients and edge servers during the intermediate steps of the training phase. In the next section, we describe a modification of Algorithm 2 to account for such privacy constraint.

IV. Privacy-Aware Federated Split Learning

To describe the design of our Privacy-Aware Federated Split Learning, we first detail the privacy attacker model and then explain the drawbacks of existing privacy-preserving methods designed for distributed learning architectures [21]. In the rest of this paper, we denote the machine learning models as *learner* and the adversary as *attacker*.

Algorithm 1 Parallel Split Learning [3] (PSL)

procedure PSL($Loss\_func_s$)
  // split layers in clients and server.
  // randomly initialize weights of layers.
  $results$ = array of $Interm\_r_i$
  $gradient'$ = array of gradients
  for $epoch$ in $epochs$ **do**:
    in parallel for $client_i \in clients$ **do**
      $Interm\_r_i \leftarrow F_{c,i}(B_i)$
      $Send((Interm\_r_i, labels_i), Server)$
      $results.append(Interm\_r_i)$
    end for
    $output \leftarrow F_s(results)$
    $Loss \leftarrow Loss\_func_s(output, labels)$
    $gradient \leftarrow G(Loss)$
    $gradient' \leftarrow F_s^T(gradient)$
    in parallel for $client_i \in clients$ **do**
      $Send(gradient'_i, client_i)$
      $F_{c,i}^T(gradient'_i)$
    end for
  end for
end procedure

Algorithm 2 Privacy-Oblivious Federated Split Learning (FSL)

procedure FSL($Loss\_func$)
  // split layers in clients and edge servers.
  // randomly initialize weights of layers.
  for $epoch$ in $epochs$ **do**:
    in parallel for each $client_i$, $edge\_server_i$ **do**
      while $client_i$ has new data to train on **do**
        $Interm\_r_i \leftarrow F_{c,i}(B_i)$
        $Send((Interm\_r_i, labels_i), edge\_server_i)$
        $output_i \leftarrow F_{s,i}(Interm\_r_i)$
        $Loss_i \leftarrow Loss\_func_s(output_i, labels_i)$
        $gradient_i \leftarrow G(Loss_i)$
        $gradient'_i \leftarrow F_{s,i}^T(gradient_i)$
        $Send(gradient'_i, client_i)$
        $F_{c,i}^T(gradient'_i)$
      end while
    end for
    Avg(Weights layers of $edge\_servers$)
  end for
end procedure

A. Privacy Attacker Model and Assumptions

Let us consider an attack on the distributed neural network, whose goal is to reconstruct the original data from the intermediate data that a client sends to the edge server during the training phase. We assume that an adversary can intercept the unencrypted intermediate data that a client is sending to the edge server[4]. During the training phase, the attacker's goal is to reconstruct the source data based on the data observed at each intermediate step of the training. To perform such reconstruction, the attacker needs a machine learning model that learns the source data used in the training phase. An example of such a machine learning model is an encoder-decoder neural network [26]. We also assume that the architecture of the neural network owned by each client is known to the attacker. In particular, the attacker neural network is constructed in such a way to reconstruct the data, using the layers of the client(s)' neural network as encoder. We also assume that the attacker's neural network is trained on a similar dataset.

B. Our Solution: Client-Based Privacy in Distributed Setting via Distance Correlation

In this subsection, we describe our privacy-preserving distributed learning solution. We assume that both client and server processes run on different machines, and the data during the training phase is exchanged over a computer network. To overcome the limitation caused by a single global loss function and bring privacy awareness into distributed settings, we propose a new technique called *Client-Based Privacy Approach*. The intuition behind such an approach is that we want to compute two different loss functions. The first one is privacy-aware (e.g., DC and differential privacy) and runs only in clients. A second global loss function is computed on the server and propagates across both client and server during the training process. In the following text, we discuss the distance correlation loss function using the DP-SGD algorithm [22] in the clients. We named them as *Client-Based Privacy Approach via DC* and *Client-Based Privacy Approach via DP*.

Our design is inspired by the *NoPeek* approach [21] but applied in distributed settings; that is, we use a loss function called *Distance Correlation* [21], to measure how the source data is different from the intermediate data.

This approach is described in Algorithm 3. By controlling how often the DC function is minimized and the weights updated, we can tune the accuracy-privacy tradeoff.

Note how, unlike NoPeek, our solution minimizes two loss functions: the Cross-Entropy computed within the server, and the Distance Correlation, calculated in the client, independently. By increasing the frequency at which both loss functions are recomputed (Algorithm 3, line 8), we can increase privacy, increasing the distance between the intermediate results and the original data (de facto adding noise to the training process). The Distance Correlation solves an optimization problem that maximizes the diversity between the original training data and the intermediate result.

Analyzing further Algorithm 3, we note that lines 8-15 show the Cross-Entropy computation and lines 17-19 show the steps used to calculate the Distance Correlation loss. Note how we

[4]An authentication protocol [24], [25] is needed before communications between entities take place to enhance the privacy guarantee, but that is not the focus of this paper. Here we assume the adversary can bypass the authentication process and read the data flowing over the wire.

specify the learning frequency using the Distance Correlation in the client in line 8.

Algorithm 3 Privacy Aware FSL with Distance Correlation

```
procedure FSL_PA(Loss_func_s, Loss_func_c = DC)
    // split layers in clients and edge servers.
    // randomly initialize weights of layers.
    for epoch in epochs do:
        in parallel for each client_i, edge_server_i do
            while client_i has new data to train on do
                Interm_r_i ← F_{c,i}(B_i)
                if (epoch mod frequency) == 0 then
                    Send((Interm_r_i, labels_i),
                          edge_server_i)
                    output_i ← F_{s,i}(Interm_r_i)
                    Loss_i = Loss_func_s(output_i, labels_i)
                    gradient_i ← G(Loss_i)
                    gradient'_i ← F^T_{s,i}(gradient_i)
                    Send(gradient'_i, client_i)
                    F^T_{c,i}(gradient'_i)
                else
                    Loss_i ←
                          Loss_func_c(data, Interm_r_i)
                    gradient_i ← G(Loss_func)
                    F^T_{c,i}(gradient_i)
                end if
            end while
        end for
        Avg(Weights layers of edge_servers)
    end for
end procedure
```

V. Evaluation of Privacy-Oblivious Hybrid Federated-Split Learning Architectures

In this section, we analyze the performance of our proposed Federated Split Learning (FSL) architecture, compared to the existing Parallel Split Learning (PSL) [3], detailed in Figure 3[5]. Note that in our FSL architecture, each client is paired with an edge server, while in Parallel Split learning, we only have one server. In particular, we first describe the experimental setup (Subsection V-A), and then we discuss the metrics and methodology used (Subsection V-B). Finally, we show the performance evaluation in terms of training completion time (Subsection V-C1), memory usage (Subsection V-C2), and learner accuracy (Subsection V-C3).

A. Experimental Setup and Use Case

To analyze the approaches introduced in the previous section, we used a well-known machine learning application: a digits classification task, using the MNIST dataset [15] [6] with the Lenet Neural network [27]. We tested our privacy-oblivious distributed learning architecture with a simulation campaign and a prototype deployed over several bare metal machines within the Chameleon Cloud testbed [28]. Each bare metal instance reserved had 48 compute cores, 187 GB memory, and one NVIDIA RTX6000 GPU.

B. Metrics and Methodology

We compare the PSL and FSL in terms of training time evaluation (Subection V-C1). We move to an analysis of the memory consumption (Subsection V-C2), and finally, we study the learner accuracy of these two architectures (Subsection V-C3).

Empirically, we have determined that at the 20^{th} epoch, both distributed hybrid architectures PSL and FSL with three different numbers of clients converge (Figure 5), i.e., after that, each additional epoch enhances the learner accuracy less than 1%. Based on this observation, we define some metrics that are used in the following section.

We define the *training time* as the time needed to compute 20 epochs of training by a client or a server.

By *memory consumption*, we mean the maximum (GPU) memory allocated during the training process.

Finally, we plot the *learner accuracy* of both architectures, varying the number of client and server pairs.

We evaluate the performance of both architectures by varying the number of clients from 20 to 500 and changing the available data at each client.

The first two subsections show our results under a data partition strategy designed to isolate the advantage of FSL over PSL only thanks to the parallelism of pairs of client and edge server. The expectation is that with more clients and fewer data in each client, both architectures would reach a lower learner accuracy; we want to assess how severe such degradation could be, so we evaluate the training time and the memory utilization (Figure 4).

C. Evaluation Results

1) Training Time Evaluation: In our *privacy-oblivious* setup, we evaluated the performance in terms of training time. In particular, in Figure 4 (left), we can see how the training time of FSL for the (edge) servers (during the 20 epochs) is shorter, and so better, than PSL for a different number of clients.

Take-home message. *The training time in our model is also consistently shorter at the client-side, compared to the* Parallel Split learning *architecture. We consider the reason to be the overall more severe resource contention in the* PSL *architecture.*

Those two observations show that the FSL can better parallelize training jobs among clients and edge servers. The FSL architecture can be especially advantageous as the number of clients increases. In the PSL architecture, the single server could become a significant bottleneck during training.

[5]We did initially consider the case of a single ML process on a single machine to ensure that the ML model converges before distributing the computation, but this case is not the focus of this paper.

[6]In this paper, we show results for only one dataset due to lack of space.

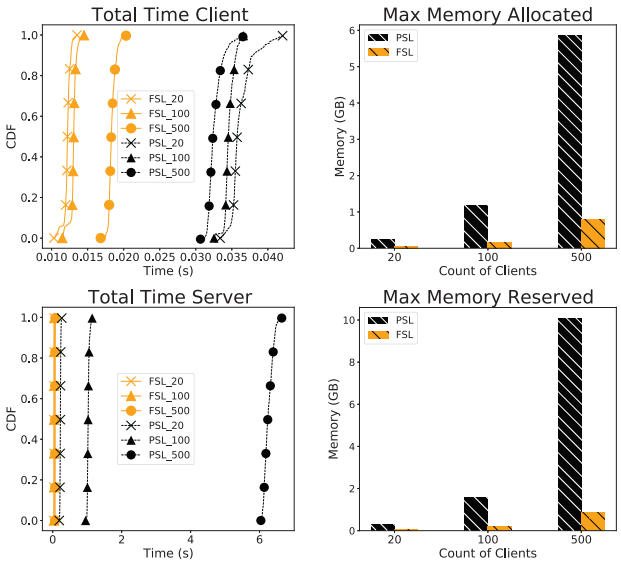

Fig. 4: Left: client's and server's total time at each epoch; Right: Maximum reserved and allocated GPU memory. Note: we randomly divided the full dataset to 500 partitions and for each configuration of 20, 100 and 500 clients, we pick the 20, 100 or 500 from the 500 partitions, respectively.

2) Memory Consumption Evaluation: In this experiment, we measure the maximum GPU memory reserved during the training process for both processes, those running on the clients and those running on the servers of the distributed split architectures (Figure 4 Right). One important implementation detail is that if some memory has been reserved by the Caching Allocator of the PyTorch library, it is not available for other GPU applications. This is why we use the Caching Allocator to measure the memory demand of our architectures.

Take-home message *Comparing the memory usage under a different number of clients, we note how the maximum memory allocated in our FSL is lower than that in PSL.* We also note that in *Parallel Split Learning*, the maximum memory allocated or reserved is reached when all the intermediate data from clients stay in the server queue waiting to be processed. Such memory value depends on how many clients we have in the system, the size of the gradients, and the computational graph. Our analysis refers to a worst case, but on average, the workloads may not be bounded by the available memory.

Note also that, since adding more clients impacts the propagation graph of the training process, we expect to see a larger memory increase as the number of clients grows. We conclude that FSL is more scalable than PSL since it consumes less memory when there are more clients.

3) Learner Accuracy Evaluation: In the previous subsections, we have shown how FSL is faster than PSL, while, at the same time, it has a lower worst-case memory utilization. This is because FSL can parallelize the training jobs better than PSL. We now show a tradeoff analysis obtained from comparing the two architectures.

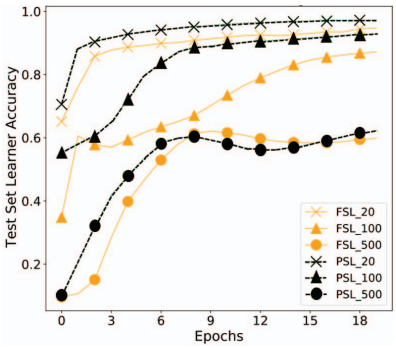

Fig. 5: Learner Accuracy (privacy oblivious) with different number of clients. We randomly divided the full dataset to 20, 100 and 500 partitions, for each configuration of 20, 100 and 500 clients respectively.

Consider Figure 5. We evaluate the accuracy of both architectures under a different number of clients. Given the Neural Networks and the datasets that we used, both FSL and PSL have high accuracy (from 87% to 93%) as long as each client has enough data to train the machine learning model. In our experiments, we used $60,000$ images evenly distributed across 20, 100, or 500 clients. The data used for training was Independent and Identically Distributed (IID) across the clients.

With a fixed training dataset of $60,000$ images, we can see a significant drop in the test-set accuracy in both architectures. While with 20 or 100 clients, the accuracy of PSL is slightly higher than FSL, with 500 clients, the accuracy is equivalent. This result shows that there is a tradeoff between accuracy and resource demand. The reason for such lower accuracy of FSL over PSL is that the neural network weight averaging step of our FSL, unlike FedAvg in Federated Learning, only averages the weights in edge servers after each training epoch to guarantee some privacy level on the training data. There is hence a tradeoff between privacy and accuracy. In the next section, we present the results of our experiments for privacy guarantees.

VI. Experiments for Privacy-Aware Federated Split Learning

In this set of experiments, we compare the privacy-preserving properties of FSL versus PSL. By privacy, we mean the ability to reconstruct the training dataset (in our case, images) under the attacker model discussed in Section IV-A. We evaluate the privacy-preserving properties using three implementations of both FSL and PSL. The first implemented both distributed architectures on a single machine, using the PyTorch library. In such settings, we analyzed the impact of our architecture when using the NoPeek [21], a logically centralized privacy-preserving algorithm. In particular, we test the following hypothesis: *Is the NoPeek algorithm sufficient to guarantee training data privacy in a distributed learning setting?* Our second implementation was instead logically

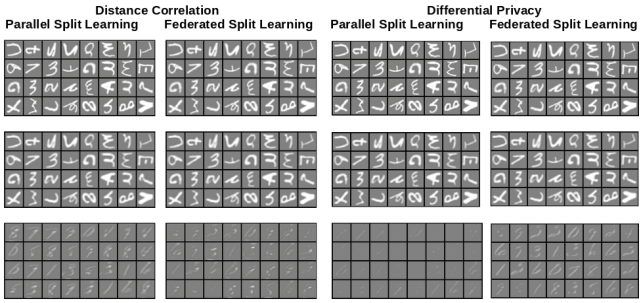

Fig. 6: *First and Second row*: batch of 32 original images and output images of the auto-encoder Neural Network. *Third row:* (poorly) reconstructed batch images.

distributed; that is, we used several clients and edge servers with processes. We implemented the Client-Based Privacy Approach algorithm, our privacy-preserving approach, using two techniques, the Distance Correlation function, and the Differential Privacy. The third implementation is a real distributed version of the second one. In all our implementations we used PyTorch, PySyft [12] and the PyGrid [13] libraries.

To evaluate the privacy-preserving properties of our two-hybrid split-federated learning architectures under observation, we used our Client-based privacy approaches. In the rest of this section, we first detail our evaluation settings and then summarize our findings.

A. Evaluation Setup

1) Training the Attacker's Auto-Encoder Neural Network: Figure 6 shows the results obtained from the reconstruction attack. We used Figure 6 to illustrate the steps of our experiments and evaluate how different attacker settings may impact the reconstruction results. To train the attacker's auto-encoder neural network, we used different random subsets of training data, producing different reconstruction results. Then, we computed the attack resilience τ, which represents the probability that the attacker misclassifies the images based on the reconstructed source data, and it is formally defined in Section VI-B.

In Figure 6, we report the results obtained with seed no. 72. Each seed specifies a random subset of data for training the attacker's NN. In the first row of images in Figure 6 we report the original batch of images at the 50^{th} epoch of the auto-encoder training. The second row shows how the output obtained from the decoder partition of the attacker's Neural Network is clear and well-reconstructed, indicating that the attacker neural network has been adequately trained. Finally, the third row shows the images (poorly) reconstructed by the attacker based on the intermediate data of the learner.

Running several experiments, we found that different seeds, i.e., different subsets of the data used for training, lead to significantly different reconstructions results. To quantify the reconstruction ability of the attacker, we measure the attack resilience τ. In the rest of this section, we detail our findings.

2) Privacy Evaluation: Experimental Settings and Methodology: To simulate a realistic use case, we train the attacker's auto-encoder on the EMNIST-letters dataset [29] that has similar features as the source MNIST dataset of the learner. The attacker is hence capable of distinguishing many but not all the features in the source dataset.

To obtain an upper bound on the attack resilience, we used a locally trained model that classifies all the reconstructed images by the attacker. This experiment aims to evaluate the frequency at which the attacker can reconstruct the source images across different training conditions.

In particular, we first train the Lenet neural network [27] with the first several layers in the client and the remaining part running at the edge server, based on the full MNIST dataset.

While the training process is running, we saved the intermediate data between clients and edge servers for each batch of data at each epoch so that our attacker neural network can try to reconstruct the original source file (that we wish to keep private) based on the saved intermediate files. Next, the attacker first trains on its auto-encoder on a subset of shuffled images from the EMNIST-letters dataset. We used an auto-encoder to reconstruct the first row in Figure 6; we show the results in the second row. We then reconstructed the source images from the intermediate data with this trained model. The third row in Figure 6 are examples of those reconstructions. Finally, we compute the predictions and verify the correctness of the reconstructed images with the trained local learner's model. The random seed has a significant impact on the reconstruction results; in the third row of Figure 6, we repeat our experiment multiple times with different seeds to compute an average accuracy that illustrates the privacy benefits of our proposed architecture augmented with our Client-Based Privacy Approach.

B. Privacy Evaluation Metrics and Parameters

The metrics that we use in this section are Learner accuracy and attack resilience. We define the *attack resilience* τ as follows:

$$\tau = 1 - \frac{\|correct\|}{\|reconstructed\|} = 1 - \frac{c}{r}, \tag{1}$$

where $c = \|correct\|$ denotes the amount of reconstructed data being correctly classified by the learner neural network, and $r = \|reconstructed\|$ denotes the amount of total reconstructed data. For example, $\tau = 0$ signifies no resilience to the attack since the reconstructed images have the same prediction result of the source images with the trained learner's model. Note that the definition of τ can be generalized to learning objectives different than classification, for example, to regression problems.

The parameters that we tuned for the experiments are DC frequency for the *Client-Based Privacy via Distance Correlation (DC)* and noise multiplier ϵ for *Client-Based Privacy via Differential Privacy (DP)* over the full learner model. The

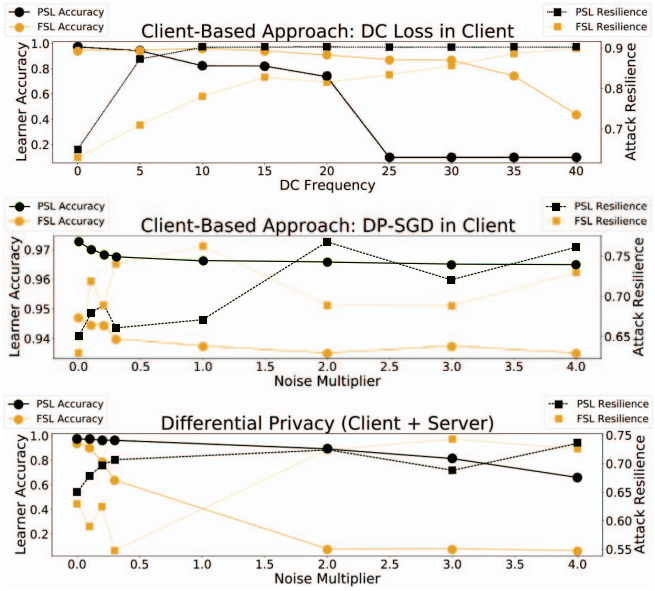

Fig. 7: Learner accuracy and attack resilience (τ) with 20 clients for Client-Based Privacy Approaches (via DC (*top*) and via DP (*middle*)) and DP-SGD on the global learner model (*bottom*). Our FSL with both client-based policies guarantee high-level of privacy and accuracy.

DC frequency indicates how often we train the client with the DC loss function. The noise multiplier is used to specify the magnitude of each Gaussian noise we added in the gradient during backward propagation (as required by the Differential Privacy algorithm).

C. Approaches

To evaluate the privacy guarantee of the architectures under consideration, we implemented them with our Client-Based Privacy via DC Loss function that alternates the Distance Correlation (DC) Loss Function with the computation of the Cross-Entropy. Then, we compare our FSL with Client-Based Privacy via DP applied only within each client. Such an algorithm, also known as DP-SGD consists of changing the direction of the Stochastic Gradient Descent using a random perturbation ϵ. Such a noise introduced into the model also introduces noise into the intermediate data so that the reconstruction of source data (i.e., images) becomes harder. We implement Client-Based Privacy via DP by replacing line 14 in Algorithm 2 with the line in Algorithm 4.

Finally, we evaluated the privacy obtained by implementing DP-SGD within the client and the server, that is, adding the noise ϵ also to the backward propagation phase.

Algorithm 4 Client-Based-DP-Backward-Propagation

1: $F^{T}_{c,i}(gradient', \epsilon)$

D. Privacy Evaluation Results

1) Privacy Evaluation using NoPeek Approach: In this experiment, we evaluate the privacy of our architectures under consideration using the NoPeek algorithm [21] within our local implementation of both FSL and PSL architectures. The results are shown in Table II. With the NoPeek method, we can see how we obtain a very high level of privacy regardless of the used learning architecture (both FSL and PSL architectures reach higher than 95% learner accuracy). Moreover, the attack resilience τ based on the intermediate data generated with the two models is above 97%. Thus, the NoPeek algorithm would give excellent attack resilience and accuracy results (a desirable outcome). However, implementing the NoPeek algorithm in real distributed settings is impractical. To do so, we would need to compute the loss function by sending the distance correlation function over the network, therefore risking a loss of privacy. Motivated by this limitation, in the next section, we tested other approaches that are more suitable for a distributed learning architecture. Table II only shows the effectiveness of the Distance Correlation loss function used in NoPeek, which we then adopt to realize our distributed client-based privacy approach.

cases	PSL	FSL
attack_resilience(τ)	0.9733	0.9837
learner_accuracy	0.9702	0.9614

TABLE II: NoPeek stats with 20 clients after finishing 20^{th} epoch of learner.

2) Evaluation Result using Client-Based Privacy via Distance Correlation: Consider Figure 7 (top). The left y-axis shows the learner's classification accuracy, and the right y-axis shows the attack resilience. The x-axis indicates the DC frequencies used. Notice that a value of $DC\_frequency = 0$ signifies that a DC loss was never computed.

From the experiments in Figure 7 (top), we note that the attack resilience grows with the DC frequency. Note also how the trends of PSL and FSL architectures are different. When the DC Frequency increases from 10 to 20, FSL has a higher learner accuracy while the attack resilience is comparable to the one achieved by PSL. In such DC frequency range, FSL keeps a learner accuracy at about 95%, while the accuracy of PSL decreases to about 80%. Although the attack resilience of PSL reaches 0.9, which means 90% of the reconstructed images were incorrectly classified by the attacker, we notice that at the same time, the learner accuracy drops to zero. This means that the PSL cannot classify the images anymore because we make the intermediate data different from the source data. So the intermediate data lost the essential features necessary for the classification. We can see a similar behavior for the FSL algorithm when we increase the frequency to 40. However, we note that FSL gives us a more comprehensive range of DC frequency to get both good learner accuracy and good attack resilience, i.e., FSL beats PSL in DC frequency range of $[10, 35]$, while PSL beats FSL only at DC frequency

of 5.

Take-home message. *We conclude that FSL augmented with our Client-Based Privacy Approach via DC can reach a higher learner accuracy and provide good privacy guarantee more stably than that of PSL.*

The reason is that FSL is more stable: in PSL, we have only one server, and so during the Learner training, PSL computes only one loss function. The weights in the server's neural network would be affected by the changes in clients' weights caused by Distance Correlation epochs. This means that the Cross-Entropy loss would consider the sum of all changes introduced by DC in different clients. On the contrary, in FSL, we average all the loss functions computed by each edge server. So each Cross-Entropy loss in the edge server only considers the changes introduced by Distance Correlation in the pairing client. So, FSL is more stable and easy to tune. On the other hand, PSL is prone to noise introduced by DC loss, i.e., the accuracy of PSL can drop to zero quickly if we do not carefully tune the parameters to overcome this disadvantage.

3) Privacy Evaluation using the Differential Privacy Approach: In this subsection, we evaluate our Client-Based Privacy Approach via DP instead of DC loss and compare it with a commonly used DP-SGD algorithm [22] used in the full learner model.

We summarize the results in Figure 7 (middle and bottom). As in our prior experiments, the left y-axis shows the learner's classification accuracy, and the right y-axis shows the attack resilience. The x-axis shows the noise multiplier, the ϵ parameter, representing how much noise we add to the gradients during the training process. This noise is added to randomly change the direction of the gradient descent to preserve source data privacy. A value of zero on the x-axis indicates that the Differential Privacy algorithm is not used in that experiment. Figure 7 (middle) shows the experiments where the Differential Privacy algorithm is applied in the clients only. We can see that the learner accuracy is high and similar to the case without privacy approaches, while the attack resilience is a little higher than the privacy-oblivious case.

Comparing with PSL, for most data points, FSL has similar learner accuracy and attack resilience. Also, the level of attack resilience reached with our Client-Based Privacy Approach via DP is lower than the one in the Client-Based Privacy Approach via DC for both architectures. The difference is that Differential Privacy adds random noise to the client's gradients. In contrast, Distance Correlation adds noise in the gradients' direction to maximize the difference between source data and intermediate data. In this way, the latter approach can guide the descent of the gradients in a better way, maximizing an optimization function.

Take-home message. *To conclude, we have found that Client-Based Privacy Approaches can keep source data private, regardless of the distributed learning architecture employed.*

We now compare the privacy guarantee when applying DP-SGD in the global learner — Figure 7 (bottom). We note that FSL does not work well with this privacy approach. The learner accuracy drops to about zero for the same noise multipliers used in the previous experiment. This set of experiments show a drawback of our FSL architecture. The averaging step would balance out the random noise introduced to weights in the edge servers by the gradients during the training epochs. Notice that unlike FedAvg in federated learning, our averaging step only averages the weights in the edge servers and leaves the client's weights intact for privacy-preserving purposes. So, the resulting model may not be in the converged state right after the averaging step, especially when applying a bigger noise multiplier. In our FSL architecture, we do not want to make the weights too different among edge servers in general (either the differences come from source data or random noise), and modifying the weights in small client's models is more efficient. Furthermore, this discussion also illustrates why FSL consistently achieves a little lower learner accuracy than PSL in all experiments.

Take-home message. *To conclude, we can see that the previously described Client-Based Privacy Approach via DC can reach better performance in terms of both privacy and accuracy for both the architectures than the Client-Based Privacy Approach via DP. Also, our Client-Based Privacy Approach is the only efficient algorithm to keep source data private in FSL architectures.*

VII. Conclusion

Recently, distributed machine learning techniques such as Federated and Split Learning have been proposed in isolation to better scale machine learning jobs and to preserve user and data privacy. Both of these distributed learning architectures have merits but also drawbacks. In this paper, we design a hybrid Federated Split Learning architecture that can get the best of both while limiting the drawbacks. We extensively evaluated our proposed architecture with a tradeoff analysis with simulations and a prototype evaluated over GPU-based clouds. In particular, we assessed the learner accuracy, latency, memory performance, and privacy guarantee of Federated Split Learning, comparing it to a recently proposed hybrid architecture called Parallel Split Learning. Our result shows how our proposal is more efficient in terms of latency and memory utilization. We also evaluated several privacy policies and concluded that under our Client-Based Privacy Approach, the Distance Correlation loss function achieves better privacy regardless of the learning architecture used. Furthermore, we assessed how, in some specific setups and prediction accuracy, Differential Privacy (DP-SGD) could be a valid policy to guarantee the accuracy of the learning model and resilience to privacy attacks.

Acknowledgements

This work has been supported by National Science Foundation Awards CNS-1908574 and CNS-1908677.

REFERENCES

[1] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A distributed framework for emerging ai applications," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. USA: USENIX Association, 2018, p. 561–577.

[2] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, August 2018.

[3] J. Jeon and J. Kim, "Privacy-sensitive parallel split learning," in *2020 International Conference on Information Networking (ICOIN)*, 2020, pp. 7–9.

[4] H. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2017.

[5] F. Beaufays, K. Rao, R. Mathews, and S. Ramaswamy, "Federated learning for emoji prediction in a mobile keyboard," 2019. [Online]. Available: https://arxiv.org/abs/1906.04329

[6] A. Hard, C. M. Kiddon, D. Ramage, F. Beaufays, H. Eichner, K. Rao, R. Mathews, and S. Augenstein, "Federated learning for mobile keyboard prediction," 2018. [Online]. Available: https://arxiv.org/abs/1811.03604

[7] D. Leroy, A. Coucke, T. Lavril, T. Gisselbrecht, and J. Dureau, "Federated learning for keyword spotting," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 6341–6345.

[8] C. Park, D. Hong, and C. Seo, "An attack-based evaluation method for differentially private learning against model inversion attack," *IEEE Access*, vol. 7, pp. 124 988–124 999, 2019.

[9] K. Hsieh, A. Phanishayee, O. Mutlu, and P. Gibbons, "The non-IID data quagmire of decentralized machine learning," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 4387–4398. [Online]. Available: http://proceedings.mlr.press/v119/hsieh20a.html

[10] Y. Gao, M. Kim, S. Abuadbba, Y. Kim, C. Thapa, K. Kim, S. A. Camtep, H. Kim, and S. Nepal, "End-to-end evaluation of federated learning and split learning for internet of things," in *2020 International Symposium on Reliable Distributed Systems (SRDS)*, 2020, pp. 91–100.

[11] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[12] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," 11 2018.

[13] (2020, oct) Pygrid. [Online]. Available: https://github.com/OpenMined/PyGrid

[14] A. Abedi and S. S. Khan, "Fedsl: Federated split learning on distributed sequential data in recurrent neural networks," 2020.

[15] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[16] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1322–1333. [Online]. Available: https://doi.org/10.1145/2810103.2813677

[17] S. Hidano, T. Murakami, S. Katsumata, S. Kiyomoto, and G. Hanaoka, "Model inversion attacks for prediction systems: Without knowledge of non-sensitive attributes," in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, 2017, pp. 115–11 509.

[18] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li, and D. Song, "The secret revealer: Generative model-inversion attacks against deep neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[19] C. Thapa, M. A. P. Chamikara, and S. Camtepe, "Splitfed: When federated learning meets split learning," 2020.

[20] A. Abedi and S. S. Khan, "Fedsl: Federated split learning on distributed sequential data in recurrent neural networks," 2020.

[21] P. Vepakomma, A. Singh, O. Gupta, and R. Raskar, "Nopeek: Information leakage reduction to share activations in distributed deep learning," in *2020 International Conference on Data Mining Workshops (ICDMW)*, 2020, pp. 933–942.

[22] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 308–318. [Online]. Available: https://doi.org/10.1145/2976749.2978318

[23] (2021, oct) Opacus. [Online]. Available: https://opacus.ai/

[24] S. Qiu, D. Wang, G. Xu, and S. Kumari, "Practical and provably secure three-factor authentication protocol based on extended chaotic-maps for mobile lightweight devices," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2020.

[25] Q. Jiang, N. Zhang, J. Ni, J. Ma, X. Ma, and K.-K. R. Choo, "Unified biometric privacy preserving three-factor authentication and key agreement for cloud-assisted autonomous vehicles," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 9, pp. 9390–9401, 2020.

[26] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length and helmholtz free energy," in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, ser. NIPS'93. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, p. 3–10.

[27] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[28] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, J. Mambretti, A. Barnes, F. Halbach, A. Rocha, and J. Stubbs, "Lessons learned from the chameleon testbed," in *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association, July 2020.

[29] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "Emnist: Extending mnist to handwritten letters," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 2921–2926.